

General Computation of Transition Adjacency Relations Based on Complete Prefix Unfolding

Jisheng Pei, Lijie Wen, and Xiaojun Ye

School of Software, Tsinghua University, Beijing 100084, P.R. China
 pjs07@mails.thu.edu.cn, {wenlj, yexj}@tsinghua.edu.cn

Abstract. An increasing number of works have devoted to the application of Transition Adjacency Relation (TAR) as a means to capture behavioral features of business process models. In this paper, we systematically study the efficient TAR derivation from process models using unfolding technique which previously has been used to address the state space explosion when dealing with concurrent behaviors of a Petri net. We reveal and formally describe the equivalence between TAR and Event Adjacency Relation (EAR), the manifestation of TAR in the Complete Prefix Unfolding (CPU) of a Petri net. By computing TARs from CPU using this equivalence, we can alleviate the concurrency caused state-explosion issues. Furthermore, structural boosting rules are categorized, proved and added to the TAR computing algorithm. Formal proofs of correctness and generality of CPU-based TAR computation are provided for the first time by this work, and they significantly expand the range of Petri nets from which TARs can be efficiently derived. Experiments on both industrial and synthesized process models show the effectiveness of proposed CPU-based algorithms as well as the observation that they scale well with the increase in size and concurrency of business process models.

Keywords: Business Process Model, Transition Adjacency Relation, Petri net, Complete Prefix Unfolding

1 Introduction

Business process models are having increasing values for various types of organizations, which are therefore maintaining large process model repositories. In order to optimize and analyze these models and enhance their business process management, a variety of analysis techniques are proposed, e.g. similarity based model retrieval [7], conformance checking [5], etc.

Recently, relational semantics of process models, which capture action dependencies as order dependencies for transitions in Petri nets, have attracted the attention of many researchers for their application in answering causality related analysis questions. For instance, behavioral relations have been the basis for checking conformance between a process model and the actual traces [2], to

assess the similarity of process models [7], or to manage process model variants [6]. Transition Adjacency Relation (TAR), which is based on direct successorship of transitions [1], and also referred to as footprint [8], has already been effectively applied in a wide range of business process model analysis scenarios such as conformance checking and similarity measurement [1,9,10].

As TAR has been widely applied, approaches for computing TARs are also raised and discussed as an important issue. For example, in [9], the authors observed reducing silent transitions before computing TAR not only increases the semantic relevance of the derived TARs, but also reduces the size of concurrent structures and therefore the related computational costs, as their reachability-graph based approach, although being general and applicable to all bounded nets, is highly influenced by their concurrent structures. However, after reduction, TAR computations from the reduced nets are, once again, still based on reachability graphs (RG). Therefore, the state explosion issue of RG, which is caused by net concurrency, is only partially reduced, but not really resolved. In [10], Jin et al. provided an unfolding based technique for efficient TAR derivation. But the rules used in [10] to derive TARs are only reliable in free-choice WF-nets, and tend to produce a lot of false TARs in more general classes of Petri nets, e.g. bounded nets with non-free-choice structures. On the other hand, in [11], the authors discovered that behavioral relations [2] can be both efficiently and generically computed from the Complete Prefix Unfolding (CPU) of a Petri net, from which we are inspired to ask if TAR computation can also benefit from utilizing structural relations in CPU with the same generality.

The issues reflected in the above works are calling for an approach that not only addresses the concurrency challenge, but also preserves the generality and correctness of the solution (e.g. for non-free-choice nets). In this paper, after formally introducing the relations between TARs and Event Adjacency Relations (EARs), their manifestations in the CPU of the original Petri net, we realize that TARs can be both generally and efficiently computed from a wide range of Petri net systems (i.e. bounded nets), and a general CPU-based TAR computation algorithm that applies to all bounded nets is proposed. Based on this general algorithm, additional structural insights can be added to obtain more efficient TAR computing algorithms. For example, we proved that *co* relation between events implies TAR see Prop. 5 of Section 4). What is more, we refine and further develop the structural rules used by [10] in free-choice WF-nets into two novel structural concepts, i.e. "Max-Event Adjacent" (for bounded nets) and "Max-event Adjacent By-Jump", which can be applied in non-free-choice nets with formally proved correctness. Based on these concepts, we present an improved version of the general CPU-based TAR computing algorithm that may cost only a fraction of the overhead taken by the general algorithm. Thus, we significantly expanded the range of net system classes whose TARs can be efficiently computed using CPU.

We organize the rest of our paper as follows. In Section 2, we recall some definitions of Petri net and CPU, as well as the definition of TAR. Section 3 formally analyzes how TAR is manifested in CPU and reports a general and

efficient algorithm for computing TAR of bounded nets from CPU. Section 4 presents the improved TAR computing algorithm based on structural insights of concurrency and Max-Event Adjacency. Section 5 evaluate the effectiveness and scalability of our algorithms by comparative experiments. Section 6 concludes our work and points out several interesting future directions.

2 Preliminaries

Definition 1 (Petri net [13]). A Petri net is a triple $N = (P, T, F)$, where P and T are finite set of places and transitions respectively ($P \cap T = \emptyset$ and $P \cup T \neq \emptyset$), and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation). Denote $X = P \cup T$, for a node $x \in X$, $\bullet x = \{y \in X | (y, x) \in F\}$, $x \bullet = \{y \in X | (x, y) \in F\}$.

Definition 2 (Petri net semantics). Let $N = (P, T, F)$ be a Petri net, then

- $M: P \rightarrow \mathbb{N}$ is a marking of N , \mathbb{N} is the set of non-negative integers. \mathbb{M} denotes all possible markings of N . $M(p)$ denotes the number of tokens in place p .
- For $\forall t \in T, M \in \mathbb{M}$, t is enabled in M , denoted by $(N, M)[t]$, iff $\forall p \in \bullet t : M(p) \geq 1$.
- Marking M' is reachable from M by firing t , denoted by $(N, M)[t](N, M')$, such that $M' = M - \bullet t + t \bullet$.
- A firing sequence $\sigma = t_1, \dots, t_n$ leads N from marking M_0 to marking M_n such that $(N, M_0)[t_1](N, M_1)[t_2](N, M_2) \dots (N, M_{n-1})[t_n]M_n$.
- For $\forall M, M' \in \mathbb{M}$, M' is reachable from M in N , denoted by $M' \in [N, M]$, iff there exists a firing sequence σ leading from M to M' .
- A net system is a pair $S = (N, M_0)$, where N is a net and M_0 is the initial marking of N .

Definition 3 (Transition Adjacency Relation). Let $S = (N, M_0)$ be a net system. Let t_1, t_2 be two transitions of S . We say that t_1, t_2 are in transition adjacency relation, denoted as $t_1 <_{tar} t_2$, if there exist a reachable marking M_s of S , where t_1 is enabled, and $M_s \xrightarrow{t_1} M'_s$, such that t_2 is enabled at M'_s .

The Unfolding of a Petri net is derived from its related Occurrence Net, whose definition is based on the following concepts of *causal*, *conflict*, and *concurrency* relations between nodes of a Petri net [4]:

- Two nodes x and y are in *causal* relation, denoted by $x < y$, if the net contains a path with at least one arc leading from x to y .
- x and y are in *conflict* relation, or just in conflict, denoted by $x \# y$, if the net contains two paths $s \cdot t_1 \dots x_1$ and $s \cdot t_2 \dots x_2$ starting at the same place s , and such that $t_1 \neq t_2$. In words, x_1 and x_2 are in conflict if the net contains two paths leading to x_1 and x_2 which start at the same place and immediately diverge (although later on they can converge again).
- x and y are in *concurrency* relation, denoted by $x \text{ co } y$, if neither $x < y$ nor $y < x$ nor $x \# y$.

Definition 4 (Occurrence net (of a Petri net)). An occurrence net is a net $O = (C, E, G)$, in which places are called conditions (C), transitions are called events (E), such that:

- O is acyclic, or equivalently, the causal relation is a partial order.
- $\forall c \in C : |\bullet c| \leq 1$, and for all $x \in C \cup E$ it holds $\neg(x\#x)$ and the set $\{y \in C \cup E | y < x\}$ is finite.
- For an occurrence net $O = (C, E, G)$, $\text{Min}(O)$ denotes the set of minimal elements of $C \cup E$ with respect to $<$.

The relation between a net system $S = (N, M_0)$ with $N = \{P, T, F\}$ and an occurrence net $O = (C, E, G)$ is defined as a homomorphism $h : C \cup E \mapsto P \cup T$ such that $h(C) \subseteq P$ and $h(E) \subseteq T$.

A branching process of $S = (N, M_0)$ is a tuple $\pi = (O, h)$ with $O = (C, E, G)$ being an occurrence net and h being a homomorphism from O to S . The maximal branching process of S is called unfolding [4]. The unfolding of a net system can be truncated once all markings of the original net system and all enabled transitions are represented. This yields the complete prefix unfolding.

Definition 5 (Complete Prefix Unfolding). Let $S = (N, M_0)$ be a system and $\pi = (O, h)$ a branching process with $N = (P, T, F)$ and $O = (C, E, G)$.

- A set of events $E' \subseteq E$ is a configuration, iff $\forall e, f \in E' : \neg(e\#f)$ and $\forall e \in E' : f < e \Rightarrow f \in E'$. The local configuration $[e]$ for an event $e \in E$ is defined as $\{x \in E | x < e \vee x = e\}$.
- A set of conditions $X \subseteq C$ is called co-set, iff for all distinct $c_1, c_2 \in X$ it holds $c_1 \text{ co } c_2$. If X' is maximal w.r.t. set inclusion, it is called a cut. For a finite configuration E' , $\text{Cut}(E') = (\text{Min}(O) \cup E' \bullet) \setminus \bullet E'$ is a cut, while $h(\text{Cut}(E'))$ is a reachable marking of S , denoted as $\text{Mark}(E')$, e.g. $\text{Mark}([e])$ is the set of conditions that are marked after all the events in $[e]$ are fired.
- An adequate order \prec is a strict well-founded partial order on local configurations such that for two events $e, f \in E$, $[e] \subset [f]$ implies $[e] \prec [f]$.
- An event e is a cut-off event if there exists another event e' such that $\text{Mark}([e]) = \text{Mark}([e'])$ and $[e'] \prec [e]$. e' is the corresponding event of e , denoted as $e' = \text{corr}(e)$.
- A Complete Prefix Unfolding (CPU) is the greatest backward closed subnet of a branching process containing no events after any cut-off event.

Example 1. Fig. 1(a),(b) and (c) illustrate a Petri net, its related CPU and reachability graph respectively. In this paper, we name each event with its corresponding transition name, which is followed by '-' and then its construction order in the CPU. For example, in Fig. 1(b), T_8-10 is an event corresponding to T_8 of the original net ($h(T_8-10) = T_8$), and it is the tenth event constructed in the CPU. In Fig. 1(b), $[T_8-10] = \{T_0-1, T_2-4, T_5-7, T_8-10\}$, $[T_9-9] = \{T_0-1, T_2-4, T_5-7, T_9-9\}$, and $\text{Mark}([T_8-10]) = \text{Mark}([T_9-9]) = \{P_1, P_3, P_{10}\}$, thus T_8-10 is a cut-off event, and T_9-9 is its corresponding event.

In contrast to a reachability graph (RG), which is prone to encounter state explosion with Petri net concurrencies (because it enumerates all the possible states of the net), the complete prefix unfolding of a Petri net is much smaller in size and takes less time to construct, as illustrated in Fig. 1(b) and 1(c).

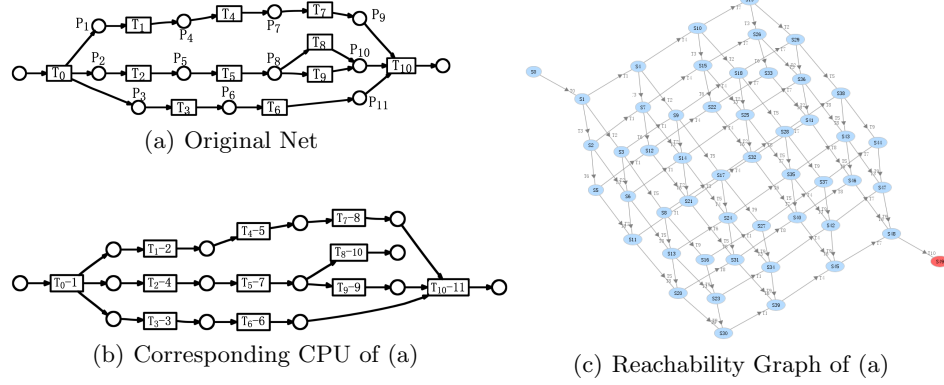


Fig. 1: A Petri net example, its related CPU and reachability graph (RG)

In this paper, we adopt Esparza’s algorithm [4] for CPU construction, which can generate more concise CPUs compared to the original technique by McMillan [3]. We also refer readers to [4] for more introduction and examples on unfolding.

3 Computation of TARs from Complete Prefix Unfolding

In order to address the above mentioned efficiency and generality problem, in this section, we provide a detailed analysis of the relation between TARs in the original net system and their manifestations in CPU. Based on the analysis, we present a general and efficient algorithm for computing TARs from CPU.

3.1 Manifestation of TAR in CPU

Intuitively, we can analogize TARs between transitions of a Petri net with the relations between events of its CPU. Take the net system as shown in Fig. 2(a) for example, we know that $T_2 <_{tar} T_3$. In its CPU (Fig. 2(b)), this is manifested as the relation between T_2-3 and T_3-5 , as we can see that from marking M_1 (circled by dashed line) of the CPU, event T_2-3 can be triggered to reach the next marking M'_1 (represented as dark dots), where T_3-5 (corresponding to transition T_3) is enabled. Fig. 2(c) and 2(d) depict another type of example. Again we have $T_2 <_{tar} T_3$, but in this example, the relation between their corresponding events (T_2-2 and T_3-4) is not so direct. In the CPU (Fig. 2(d)), after firing of *cut-off* event T_2-2 , marking M'_1 is reached, where T_3-4 is, however, not enabled. Still we notice that from the marking M_2 reached by firing T_1-1 (T_2-2 ’s corresponding event), T_3-4 is enabled. As $h(M'_1) = h(M_2)$, we can regard this as a manifestation of $T_2 <_{tar} T_3$ in CPU even though T_2-2 is a *cut-off* event.

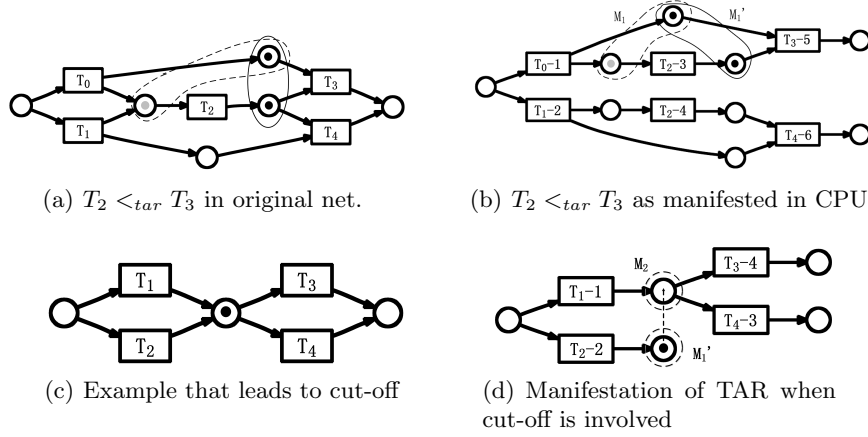


Fig. 2: Manifestations of TAR in corresponding CPU of a net

With the observation of the above two types of analogies, we notice that the counterpart relation of TAR in the CPU of a net system can be captured as:

Definition 6 (Event Adjacency Relation). Let $\pi = (O, h), O = (C, E, G)$ be a CPU of a net system $S = (N, M_0)$. Let e_1, e_2 be two events of π . We say that e_1, e_2 are in event adjacency relation (EAR), denoted as $e_1 <_{ear} e_2$, if there exist two pairs of reachable markings M_1, M'_1 and M_2, M'_2 of π such that

- $M_1 \xrightarrow{e_1} M'_1$ (e_1 is enabled at M_1)
- $M_2 \xrightarrow{e_2} M'_2$ (e_2 is enabled at M_2)
- $h(M_2) = h(M'_1)$ (note that M_2 may or may not be equal to M'_1)

Loosely speaking, $e_1 <_{ear} e_2$ implies the firing of event e_1 may immediately lead to a marking of π which relates to some marking in the original net that in turn corresponds to a marking of π that enables e_2 . With the definition of EAR, the equivalence relation between TAR and its counterpart EAR in the CPU of their net system can be described as:

Proposition 1. Let $\pi = (O, h), O = (C, E, G)$ be a complete prefix unfolding of a net system $S = (N, M_0)$. Let t_1, t_2 be two transitions of S . Then $t_1 <_{tar} t_2$ if and only if $\exists e_1, e_2 \in E : h(e_1) = t_1, h(e_2) = t_2$, and $e_1 <_{ear} e_2$.

In words, to say $t_1 <_{tar} t_2$, is equivalent to say that one can find a pair of their related events in the CPU to be in EAR, and vice versa.

Proof. (\Rightarrow) Since $t_1 <_{tar} t_2$, there exist a reachable marking M_s of S , where t_1 is enabled, and $M_s \xrightarrow{t_1} M'_s$, such that t_2 is enabled at M'_s . From $M_s \xrightarrow{t_1} M'_s$, by the properties of complete prefix unfolding we know that there exist reachable markings M_1, M'_1 of π such that for some event e_1 , $M_1 \xrightarrow{e_1} M'_1, h(e_1) = t_1$. From $M_s \xrightarrow{t_1} M'_s, M_1 \xrightarrow{e_1} M'_1, h(M_1) = M_s$ and $h(e_1) = t_1$, we can deduce that $h(M'_1) = M'_s$ (isomorphism between complete prefix unfolding and original net).

On the other hand, by applying the properties of complete prefix unfolding again, we know that there exist M_2 of π which is related to the marking M'_s of S such that $h(M_2) = M'_s = h(M'_1)$, and that an event e_2 is enabled at M_2 , that is, for some reachable marking M'_2 of π , it holds $M_2 \xrightarrow{e_2} M'_2$, $h(e_2) = t_2$. Therefore, it holds that $e_1 <_{ear} e_2$ because there exist two pairs of reachable marking M_1, M'_1 and M_2, M'_2 of π such that $M_1 \xrightarrow{e_1} M'_1$, $M_2 \xrightarrow{e_2} M'_2$ and $h(M_2) = h(M'_1)$. Thus we have proved that $\exists e_1, e_2 \in E : h(e_1) = t_1, h(e_2) = t_2$ and $e_1 <_{ear} e_2$ by constructing such e_1 and e_2 . (\Leftarrow) By the fact that $e_1 <_{ear} e_2$, there exist two pairs of reachable marking M_1, M'_1 and M_2, M'_2 of π such that $M_1 \xrightarrow{e_1} M'_1$, $M_2 \xrightarrow{e_2} M'_2$ and $h(M_2) = h(M'_1)$. By the isomorphism between π and S , we know that there exist reachable M_s, M'_s of S , such that $M_s = h(M_1)$ and $M'_s = h(M'_1) = h(M_2)$. Again, by the isomorphism between complete prefix unfolding and the original net, from the fact $h(e_1) = t_1$, $M_1 \xrightarrow{e_1} M'_1$, $M_s = h(M_1)$, and $M'_s = h(M'_1)$ we can deduce that $M_s \xrightarrow{t_1} M'_s$. Note that $M'_s = h(M'_1) = h(M_2)$, e_2 is enabled at M_2 and $h(e_2) = t_2$, so it follows that t_2 is enabled at M'_s of the original net. Thus we have for reachable marking M_s, M'_s of S , $M_s \xrightarrow{t_1} M'_s$ and that t_2 is enabled at M'_s , and therefore $t_1 <_{tar} t_2$.

3.2 EAR Based TAR Computation Algorithm

Proposition 1 guarantees that a pair of transitions of a Petri net are in TAR if and only if a pair of their corresponding events are in EAR. Next, we propose a general CPU-based TAR checking algorithm that exploits this observation to check if a given pair of transitions are in TAR. In order to better explain and prove our algorithm, we first introduce the concept of Max-Event together with three related lemmas.

Definition 7 (Max-Event Set). Let E_0 be a subset of the events in a CPU. We call the set of maximal events in E_0 with respect to causal relation as its max-event set, denoted as $Max(E_0) = \{e \in E_0 \mid \forall e' \in E_0 : e \not\prec e'\}$.

Given this definition, we have discovered the following lemmas:

Lemma 1. Let C be a configuration. Let $Max(C) = \{e_{m_1}, \dots, e_{m_k}\}$. Then

- (1) $C \setminus \{e\}$ is a configuration if and only if $e \in Max(C)$.
- (2) $[e_{m_1}] \cup \dots \cup [e_{m_k}] = C$.

Proof. (1) (\Rightarrow) Suppose to the contrary that $e \notin Max(C)$, then there is some $e' \in C$ such that $e < e'$, and thus we have $e' \in C \setminus \{e\}$ whereas $e < e'$, $e \notin C \setminus \{e\}$, so it follows that $C \setminus \{e\}$ is not a configuration, which leads to contradiction. Therefore $e \in Max(C)$. (\Leftarrow) Because C is a configuration, $\forall e, e' \in C : \neg e \# e'$. On the other hand, since C is a configuration, $\forall e' \in C : e'' < e' \Rightarrow e'' \in C$. Therefore suppose that $\exists e' \in C \setminus \{e\} : e'' < e'$ and $e'' \notin C \setminus \{e\}$, then e'' must be e , which contradicts to the fact that $e \in Max(C)$, so we must conclude that $\forall e' \in C \setminus \{e\} : e'' < e' \Rightarrow e'' \in C \setminus \{e\}$. Therefore $C \setminus \{e\}$ is a configuration.

(2) For any $e \in [e_{m_1}] \cup \dots \cup [e_{m_k}]$, if $e \in \{e_{m_1}, \dots, e_{m_k}\}$, it immediately follows that $e \in \text{Max}(C)$, and therefore $e \in C$. If otherwise, without loss of generality, suppose $e \in [e_{m_1}]$ and $e < e_{m_1}$, then as $e_{m_1} \in C$, $e < e_{m_1}$, we know that $e \in C$.

On the other hand, for any $e \in C$, if $e \in \text{Max}(C)$, then $\exists i, 1 \leq i \leq k : e = e_{m_i}$, and therefore $e \in [e_{m_1}] \cup \dots \cup [e_{m_k}]$. Otherwise, $\exists e' \in C, e < e'$, and from e' , we can thus continue to reach some e_{m_i} such that $e < e' < \dots < e_{m_i} (1 \leq i \leq k)$. Therefore $e \in [e_{m_i}]$, and consequently $e \in [e_{m_1}] \cup \dots \cup [e_{m_k}]$.

By the above two facts, we know that $[e_{m_1}] \cup \dots \cup [e_{m_k}] = C$.

Lemma 2. *Let X be a co-set in a CPU. Let $\text{Max}(\bullet X) = \{e_{m_1}, \dots, e_{m_k}\}$. Then $C = [e_{m_1}] \cup \dots \cup [e_{m_k}]$ is a configuration. And it holds that $X \subseteq \text{Cut}(C)$, $\text{Max}(C) = \text{Max}(\bullet X)$.*

Proof. We first prove that $[e_{m_1}] \cup \dots \cup [e_{m_k}]$ is a configuration. Obviously, for every $e \in C = [e_{m_1}] \cup \dots \cup [e_{m_k}]$ there exists at least one $[e_{m_i}] (1 \leq m_i \leq k)$ such that $e \in [e_{m_i}]$, and consequently $\forall e' \in C : e' < e \Rightarrow e' < e_{m_i}$ and therefore $e' \subseteq [e_{m_i}] \subseteq C$. On the other hand suppose to the contrary that there exist $e, e' \in C$ such that $e \# e'$. Similarly, there exists $[e_{m_i}], [e_{m_j}] (1 \leq i, j \leq k)$ such that $e \in [e_{m_i}]$ and $e' \in [e_{m_j}]$. if $i = j$, then we have $e, e' \in [e_{m_i}]$ and $e \# e'$, which contradicts with the fact that $[e_{m_i}]$ is a configuration. If $i \neq j$, then it follows that $e_{m_i} \# e_{m_j}$. Since $e_{m_i}, e_{m_j} \in \text{Max}(\bullet X)$, $\exists c_1, c_2 \in X : e_{m_i} < c_1, e_{m_j} < c_2$. As $e_{m_i} \# e_{m_j}$, it follows that $c_1 \# c_2$, which contradicts with fact that X is a co-set. Therefore $\forall e, e' \in C : \neg(e \# e')$.

By the above two facts, we know that $C = [e_{m_1}] \cup \dots \cup [e_{m_k}]$ is a configuration.

From $\{e_{m_1}, \dots, e_{m_k}\} = \text{Max}(\bullet X)$, we know $\forall e' \in \bullet X : e' \in C$. Therefore, for every condition $c \in X$ such that $c \notin \text{Min}(O)$, there exists one event $e' \in C$ such that $c \in e' \bullet$. As a consequence, we have $\forall c \in X : c \in \text{Min}(O) \cup C \bullet$, or equivalently $X \subseteq \text{Min}(O) \cup C \bullet$.

On the other hand, it is easy to prove that $\forall e \in C, c \in X : c \not\prec e$. Otherwise, in case there is some $e \in C, c \in X : c \not\prec e$, if $e \in \text{Max}(\bullet X)$, then for some $c' \in e \bullet : c' \in X$, we can deduce that $c < c'$, which contradicts to the fact that X is a co-set, and if $e \notin \text{Max}(\bullet X)$, then $\exists e' \in \text{Max}(\bullet X) : e < e'$, and again we can deduce that $\exists c' \in e' \bullet : c' \in X, c < e < e' < c'$, which again leads to contradiction with c co c' . Therefore, from $\forall c \in X, e' \in C, c \not\prec e'$ we can deduce $X \cap \bullet C = \emptyset$.

In summary, we know that $X \subseteq \text{Min}(O) \cup C \bullet$ and $X \cap \bullet C = \emptyset$. This is to say that $\forall c \in X : c \in (\text{Min}(O) \cup C \bullet) \setminus \bullet C$, and equivalently $X \in \text{Cut}(C) = \text{Cut}([e_{m_1}] \cup \dots \cup [e_{m_k}])$.

Finally, we prove that $\text{Max}(C) = \text{Max}(\bullet X)$. First, we prove that $\forall e \in \text{Max}(C) : e \in \text{Max}(\bullet X)$. Suppose to the contrary that there exists some $e \in \text{Max}(C) : e \notin \text{Max}(\bullet X)$. Then because e must belong to some $e_{m_i} (1 \leq i \leq k)$, and, by our assumption, $e \neq e_{m_i}$, we have $e < e_{m_i}$ and therefore $e \notin \text{Max}(C)$, which is a contradiction. Therefore $\forall e \in \text{Max}(C) : e \in \text{Max}(\bullet X)$.

Next, we prove that $\forall e \in \text{Max}(\bullet X) : e \in \text{Max}(C)$. Suppose to the contrary that there exists some $e_{m_i} \in \text{Max}(\bullet X) (1 \leq i \leq k) : e_{m_i} \notin \text{Max}(C)$. Then there must exist some $e \in C$ such that $e_{m_i} < e$. As e must belong to some $e_{m_j} (j \neq i)$,

we have $e < e_{m_j}$ or $e = e_{m_j}$. In either case, it follows that $e_{m_i} < e < e_{m_j}$. Again, since $e_{m_i}, e_{m_j} \in \text{Max}(\bullet X)$, $\exists c_1, c_2 \in X : e_{m_i} < c_1, e_{m_j} < c_2$. As $e_{m_i} < e_{m_j}$, it follows that $c_1 < c_2$, which contradicts with fact that X is a co-set. Therefore, $\forall e \in \text{Max}(\bullet X) : e \in \text{Max}(C)$.

By the above two facts, we arrive at the conclusion that $\text{Max}(C) = \text{Max}(\bullet X)$.

Lemma 3. $e \in \text{Max}(C) \Rightarrow \bullet e \subseteq \text{Cut}(C \setminus \{e\})$

Proof. The proof of this lemma is similarly to the one of Lemma 2. Let $C' = C \setminus \{e\}$. By $e \in \text{Max}(C)$, it holds that $\forall e' \in \bullet(\bullet e) : e' \in C'$. Therefore $\bullet e \subseteq \text{Min}(O) \cup C' \bullet$. On the other hand it holds that $\bullet C' \cap \bullet e = \emptyset$ (Otherwise, there is some $e' \in C'$ such that $\bullet e' \cap \bullet e \neq \emptyset$, from which it follows that $e' \# e$, which contradicts with the fact that C is a configuration).

In summary, we know that $\bullet e \subseteq \text{Min}(O) \cup C' \bullet$ and $\bullet e \cap \bullet C' = \emptyset$. Therefore $\bullet e \subseteq (\text{Min}(O) \cup C' \bullet) \setminus \bullet C'$, and equivalently, $\bullet e \subseteq \text{Cut}(C \setminus \{e\})$.

Using the above lemmas, together with the previously proved TAR-EAR relationship (Prop. 1), we can prove that:

Proposition 2. Let $\pi = (O, h)$, $O = (C, E, G)$ be the CPU of a net system $S = (N, M_0)$. Let t_1, t_2 be two transitions of S . Then $t_1 <_{\text{tar}} t_2$ iff there exists a co-set X in π such that (1) $\exists e_1 \in E : h(e_1) = t_1, e_1 \bullet \subseteq X$ and (2) $\bullet t_2 \subseteq h(X)$.

Proof. (\Rightarrow) The sufficiency is guaranteed by TAR-EAR relationship. Given $t_1 <_{\text{tar}} t_2$, by Proposition 1, we know that there exist $e_1, e_2 : h(e_1) = t_1, h(e_2) = t_2, e_1 <_{\text{ear}} e_2$. By the definition of EAR, there is a reachable marking M_1 of π such that $M_1 \xrightarrow{e_1} M'_1$, $h(e_1) = t_1$ and $h(M'_1)$ enables t_2 . Let C be the configuration fired to reach marking M'_1 and let $X = \text{Cut}(C)$ (note that X is a co-set, because $\text{Cut}(C)$ is a cut). Since t_2 is enabled at $h(M'_1)$, we know that $\bullet t_2 \subseteq h(X)$. And since $M_1 \xrightarrow{e_1} M'_1$, we have $e_1 \bullet \subseteq \text{Cut}(C) = X$.

(\Leftarrow) As for necessity, suppose there exist a co-set X such that $\bullet t_2 \subseteq h(X)$ and for some $e_1 : e_1 \bullet \subseteq X$. From $e_1 \bullet \subseteq X$, it can be deduced that $e_1 \in \text{Max}(\bullet X)$ (Otherwise, suppose to the contrary that $\exists f \in \bullet X : e < f$. Then because $e \bullet \subseteq X, f \in \bullet X$, we have $\exists c_{e_1}, c_f \in X : c_{e_1} \in e_1 \bullet, c_f \in f \bullet$ and $c_{e_1} < c_f$, which contradicts to the fact that X is a co-set). Let $\text{Max}(\bullet X) = \{e_{m_1}, \dots, e_{m_k}\} (k \geq 1)$ and, without loss of generality, let $e_{m_1} = e_1$. By Lemma 2, for configuration $C = [e_{m_1}] \cup \dots \cup [e_{m_k}]$, we have $X \subseteq \text{Cut}(C)$ and $e_1 = e_{m_1} \in \text{Max}(C)$. By (2) of Lemma 1 and Lemma 3, we know that $C' = C \setminus \{e_1\}$ is also a configuration and that $\bullet e_1 \subseteq \text{Cut}(C')$. Therefore, let M, M' be the marking of π where the conditions in $\text{Cut}(C')$ and $\text{Cut}(C)$ have tokens in them respectively, then we have $M \xrightarrow{e_1} M'$, and consequently $h(M) \xrightarrow{t_1} h(M')$. As $\bullet t_2 \subseteq h(X) \subseteq h(\text{Cut}(C))$, we know that $h(M')$ enables t_2 , which yields that $t_1 <_{\text{tar}} t_2$.

By Prop. 2, the problem of whether $t_1 <_{\text{tar}} t_2$ is transformed into the problem of whether there exists a co-set X in CPU such that $\bullet t_2 \subseteq h(X)$ and for some event $e_1 : h(e_1) = t_1$ it holds that $e_1 \bullet \subseteq X$. Apparently, if such a co-set does exist, we can build it using only the conditions that corresponds to $t_1 \bullet$ and $\bullet t_2$ in CPU.

Consequently such *co*-set must be included in one of the *cuts* (maximal co-sets w.r.t. set inclusion) w.r.t. the set of such conditions. As there are only a few such corresponding conditions for a single pair of transitions, we can check if $t_1 <_{tar} t_2$ simply by enumerating through the cuts formed by these conditions, and check if for any cut X' it holds that $\bullet t_2 \subseteq h(X')$ and for some $e_1 : h(e_1) = t_1, e_1 \bullet \subseteq X'$, which is a sufficient and necessary condition to conclude $t_1 <_{tar} t_2$.

The following Algorithm 1 adopts the above ideas to check if $t_1 <_{tar} t_2$ holds for a pair of transitions. To check condition (1) of Prop. 2, Algorithm 1 restricts the set of candidate conditions according to each event $e_1 : h(e_1) = t_1$ as $B_{prec} = \{c \mid c \text{ co } e_1, h(c) \in \bullet t_2\} \cup e_1 \bullet$ (Line 3 ~ 6). Next, the set Q of all cuts in B_{prec} are generated (Line 7). To check condition (2) of Prop. 2, each cut X'_i in Q is checked to see if $t_2 \subseteq h(X'_i)$ (Line 8 ~ 9), which, as will be proved in the following Prop. 3, is sufficient and necessary to conclude $t_1 <_{tar} t_2$. Because the size of B_{prec} and the number of events related to t_1, t_2 can generally be treated as a constant w.r.t. the growth of net size, it takes constant time overhead for Algorithm 1 to determine if $t_1 <_{tar} t_2$ for a pair of transition t_1, t_2 using CPU.

Algorithm 1 Check for TAR by Enumerating EAR Related Cuts

```

1: function CHECKBYCUTS( $t_1, t_2$ )
2:   for each  $e_1 : h(e_1) = t_1$  do
3:      $B_{prec} \leftarrow e_1 \bullet$ 
4:     for each  $t \in \bullet(\bullet t_2)$  do
5:       for each  $c \in e \bullet, h(e) = t$  and  $h(c) \in \bullet t_2$  do
6:         If  $c \text{ co } e_1$  Then  $B_{prec} \leftarrow B_{prec} \cup \{c\}$ 
7:        $Q \leftarrow$  Set of cuts in  $B_{prec}$ 
8:       for each  $X'_i \in Q$  do
9:         If  $\bullet t_2 \subseteq h(X'_i)$  Then return true
10:    end for
11:    return false
12: end function

```

Proposition 3. *Algorithm 1 returns true if and only if $t_1 <_{tar} t_2$.*

Proof. (\Rightarrow) If Algorithm 1 returns *true*, then there is a certain cut X'_i such that $\bullet t_2 \subseteq X'_i$. Also, we know that for some event $e_1 : h(e_1) = t_1, e_1 \bullet \subseteq B_{prec}$. Because conditions in B_{prec} can only be added by Line 3 or Line 6 of Algorithm 1, we know that $\forall c \in B_{prec} : (c \text{ co } e_1) \vee (c \in e_1 \bullet)$. As $X'_i \subseteq B_{prec}$, we also have $\forall c \in X'_i : (c \text{ co } e_1) \vee (c \in e_1 \bullet)$. Therefore, since X'_i is a *cut*, i.e. the maximal co-set w.r.t. set inclusion, it must hold that $e_1 \bullet \subseteq X'_i$ (otherwise, if $e_1 \bullet \not\subseteq X'_i$, conditions in $e_1 \bullet \setminus X'_i$ can then be added into X'_i to form a larger co-set, which contradicts with the fact that *cut* X is maximal). From $\bullet t_2 \subseteq X'_i$ and $\exists e_1 : h(e_1) = t_1, e_1 \bullet \subseteq X'_i$ we can conclude by Prop. 2 that $t_1 <_{tar} t_2$.

(\Leftarrow) By Proposition 2, we know that if $t_1 <_{tar} t_2$, there must exist a co-set X such that $\bullet t_2 \subseteq h(X)$, and for some event e'_1 , it holds that $e'_1 \bullet \subseteq X$. Let $B_{t_2} = \{c \in X \mid h(c) \in \bullet t_2\}$. In the loop starting from Line 2, consider the cycle when $e_1 = e'_1$, since in this cycle the loop from Line 4~6 enumerates all conditions corresponding to the $\bullet t_2$ that are in *co* with e'_1 , $B_{t_2} \cup e'_1 \bullet \subseteq B_{prec}$

holds at Line 7. Note that as $B_{t_2} \cup e'_1 \bullet$ is a co-set, it must be included in at least one of the *cuts* w.r.t. B_{prec} . Therefore in this cycle, for at least one $X'_i \in Q$, $\bullet t_2 \subseteq X'_i$ holds, and thus the function will eventually return *true* at Line 9.

Thus, we can obtain the CPU-based TAR computing algorithm (Algorithm 2) which invokes Algorithm 1 for each transition pair in a Petri net:

Algorithm 2 Derive All TARs from a Petri Net Using CPU (GENERAL)

```

1: function GENERAL( $S, \pi$ )
2:    $TAR \leftarrow \emptyset$ 
3:   for each  $t_1, t_2 \in S$  do
4:     if CheckByCuts( $t_1, t_2$ ) then  $TAR \leftarrow TAR \cup \{\langle t_1, t_2 \rangle\}$ 
5:   return  $TAR$ 
6: end function

```

Since it takes constant time for CheckByCuts to check if TAR exist between t_1 and t_2 , the time complexity of Algorithm 2 is $O(n^2)$, where n is the number of transitions in the net system S whose CPU π is also given.

4 Improved Algorithms Based on Structural Information

The above general algorithm can compute the TARs of any bounded net systems based on TAR-EAR relationship described in Proposition 1. Nevertheless, for bounded nets, we can leverage the structural relations in CPU, i.e. causality and concurrency, to boost the algorithm efficiency even further.

We begin our construction of derivation rules for TAR by presenting the following property of complete prefix unfolding, as it serves as the basis of several key proofs in this paper:

Proposition 4. (*Existence of Cut-off-free Configuration*) *For every reachable marking M of a Petri net system S , there exists a configuration \mathcal{C} in a corresponding complete prefix unfolding π of S such that $\text{Mark}(\mathcal{C}) = M$ and \mathcal{C} does not contain any cut-off events.*

Proof. Given a configuration \mathcal{C} , we denote by $\mathcal{C} \oplus E$ the fact that $\mathcal{C} \cup E$ is a configuration such that $\mathcal{C} \cap E = \emptyset$. We say that $\mathcal{C} \oplus E$ is an extension of \mathcal{C} , and that E is a suffix of \mathcal{C} . Let \mathcal{C}_1 be a configuration in the unfolding of S such that $\text{Mark}(\mathcal{C}_1) = M$. Suppose \mathcal{C}_1 contains some cut-off event e_1 and that $\mathcal{C}_1 = [e_1] \oplus E_1$ for some set of events E_1 . By the definition of a cut-off event, there exists a local configuration $[e_2]$ such that $[e_2] \prec [e_1]$ and $\text{Mark}([e_2]) = \text{Mark}([e_1])$. According to [4], there is a set of events isomorphic (corresponding to the same set of transitions) with E_1 , denoted as $I_1^2(E_1)$, such that $\text{Mark}([e_2] \oplus I_1^2(E_1)) = \text{Mark}(\mathcal{C}_1)$. Consider the configuration $\mathcal{C}_2 = [e_2] \oplus I_1^2(E_1)$. Since \prec is preserved by finite extensions (see [4]), we have $\mathcal{C}_2 \prec \mathcal{C}_1$. If \mathcal{C}_2 still contains any cut-off event, we can repeat the procedure and find a configuration \mathcal{C}_3 such that $\mathcal{C}_3 \prec \mathcal{C}_2$ and that $\text{Mark}(\mathcal{C}_3) = \text{Mark}(\mathcal{C}_2)$. The procedure cannot be iterated infinitely often because \prec is well-founded. Therefore, it terminates in a configuration \mathcal{C} in CFP which does not contain any cut-off event and $\text{Mark}(\mathcal{C}) = M$.

To explore these possibilities, the detection of TAR between any given pair of transitions t_1, t_2 is divided and conquered with the following two cases:

$$(1)t_1 \bullet \cap \bullet t_2 = \emptyset \quad (2)t_1 \bullet \cap \bullet t_2 \neq \emptyset$$

In both cases, there exist structural properties (as will be described in Proposition 5, 6 and 7) from which accelerating rules can be derived to boost the efficiency of TAR computation algorithm. First of all, in case (1), when $t_1 \bullet \cap \bullet t_2 = \emptyset$, we can prove the following:

Proposition 5. *Let S be a bounded net system and $\pi = (O, h), O = (C, E, G)$ its CPU. Let t_1, t_2 be two transitions of S . When $t_1 \bullet \cap \bullet t_2 = \emptyset$, we have $t_1 <_{tar} t_2 \Leftrightarrow \exists e_1, e_2 \in E : e_1 \text{ co } e_2, h(e_1) = t_1, h(e_2) = t_2$.*

Proof. (\Rightarrow) By $t_1 <_{tar} t_2$, there are markings M_s, M'_s in S such that $M_s \xrightarrow{t_1} M'_s$, and that t_2 is enabled at M'_s . Since $t_1 \bullet \cap \bullet t_2 = \emptyset$, we know that both the places of $\bullet t_1$ and $\bullet t_2$ have tokens in them at M_s . By Prop. 4, there exists a configuration C in π such that C does not contain any cut-off event and that $Mark(C) = M_s$. Therefore $\exists X_1 \subseteq Cut(C) : h(X_1) = \bullet t_1$. Since C does not contain cut-off events, there exists in π an event $e_1 : h(e_1) = t_1, \bullet e_1 = X_1$. Since t_2 is enabled at M'_s , we have $\exists X_2 \subseteq Cut(C) : h(X_2) = \bullet t_2, \bullet e_2 = X_2$ and $X_1 \cap X_2 = \emptyset$. Therefore, it holds that $e_1 \not\prec e_2$ and $\neg(e_1 \# e_2)$, and consequently, $e_1 \text{ co } e_2$. (\Leftarrow) For some $e_1, e_2 \in E : e_1 \text{ co } e_2, h(e_1) = t_1, h(e_2) = t_2$, we know both $\bullet e_1$ and $\bullet e_2$ are co-set and $\forall c_1 \in \bullet e_1, c_2 \in \bullet e_2 : c_1 \text{ co } c_2$. Therefore $\bullet e_1 \cap \bullet e_2 = \emptyset$ and $\bullet e_1 \cup \bullet e_2$ is a co-set, so there exists a configuration C in π such that $\bullet e_1 \cup \bullet e_2 \subseteq Cut(C)$. As $h(\bullet e_1) = \bullet t_1$ and $h(\bullet e_2) = \bullet t_2$, both t_1, t_2 are enabled at $Mark(C)$. And after t_1 is fired, $\bullet t_2$ is still enabled (because $\bullet e_1 \cap \bullet e_2 = \emptyset$). Therefore we have $t_1 <_{tar} t_2$.

By Proposition 5, we can reduce the checking of $t_1 <_{tar} t_2$ in case when $t_1 \bullet \cap \bullet t_2 = \emptyset$ to the examination of the existence of co relation between their related events in the CPU, which is almost an immediate operation and saves us the cost of cuts enumeration. The necessity for executing the costly cuts enumeration is now only limited to those consecutive pairs of transitions (that is, when $t_1 \bullet \cap \bullet t_2 \neq \emptyset$).

In case (2), when $t_1 \bullet \cap \bullet t_2 \neq \emptyset$, we could discover more structural boosting rules by introducing the concept of Max-Event Adjacent (MEA):

Definition 8 (Max-Event Adjacent). *For two events e_1 and e_2 in a complete prefix unfolding such that $e_1 < e_2$, e_1 is said to be Max-Event Adjacent (MEA) with e_2 , denoted as $e_1 \triangleright e_2$, if e_1 is among the max events of the pre-events of e_2 : $e_1 \in Max(\bullet(\bullet e_2))$. $e_1 \triangleright e_2$ can be defined equivalently as $\forall c \in e_1 \bullet : c < e_2 \Rightarrow c \in \bullet e_2$ (more convenient and efficient for implementation).*

Intuitively, $e_1 \triangleright e_2$ means e_1 are connected to e_2 entirely by length-1 paths (there exists no intermediate event on any path between them). For example, in Fig. 3, which shows the CPU of the net in the previous Fig. 2(a), events T_2 -3 and T_2 -4 are MEA with T_3 -5 and T_4 -6 respectively. But neither T_0 -1 nor T_1 -2 is MEA with T_3 -5 or T_4 -6, because T_2 -3 and T_2 -4 exist on one of the paths from T_0 -1 and T_1 -2 to T_3 -5 and T_4 -6 (as shown with the red colored arrows in Fig. 3).

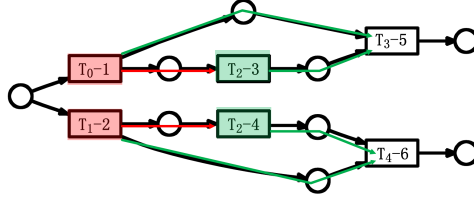


Fig. 3: An Intuitive Example of Max-Event Adjacent Structure

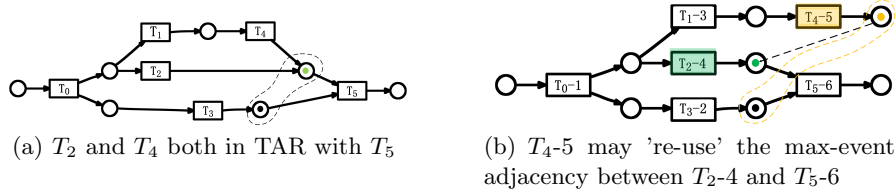


Fig. 4: Applying Max-Event Adjacent Structure across Cut-off Relation

By the above definition of Max-Event Adjacent (\triangleright), we can prove using Lemma 1 and Lemma 3 that for any bounded net's CPU:

Proposition 6. $e_1 \triangleright e_2 \Rightarrow e_1 <_{ear} e_2$.

Proof. Let $C = [e_2]$. By definition of local configuration, $Max([e_2]) = \{e_2\}$. Therefore, by Lemma 1, $C' = C \setminus \{e_2\}$ is a configuration, and by Lemma 3:

$$\bullet e_2 \subseteq Cut(C') \quad (1).$$

Moreover, it is easy to prove that $Max([e_2] \setminus \{e_2\}) \subseteq \bullet(\bullet e_2)$. From $e_1 \triangleright e_2$, we know $e_1 \in Max(\bullet(\bullet e_2))$, and therefore $e_1 \in Max([e_2] \setminus \{e_2\}) = Max(C')$. Consequently $C'' = C' \setminus \{e_1\}$ is also a configuration. And by Lemma 3 it holds that:

$$\bullet e_1 \subseteq Cut(C'') \quad (2).$$

Let the marking of π corresponding to C'' , C' and C be denoted as M_1, M'_1, M_2 respectively. By (1) and (2), it holds that $M_1 \xrightarrow{e_1} M'_1 \xrightarrow{e_2} M_2$ and $e_1 <_{ear} e_2$.

For example, in Fig. 3, from Proposition 6 we can conclude $T_{2-3} <_{ear} T_{3-5}$ and $T_{2-4} <_{ear} T_{4-6}$ and therefore $T_2 <_{tar} T_3$ and $T_2 <_{tar} T_4$ because $T_{2-3} \triangleright T_{3-5}$ and $T_{2-4} \triangleright T_{4-6}$. Moreover, we confirm that Max-Event Adjacent can be further extended and applied under certain cases even when cut-off events are involved (e.g. situation as in Fig. 4(b)), as we can prove using Lemma 3 that:

Proposition 7. Let π be the CPU of a bounded net system. Let e_1 be a cut-off event and e'_1 its corresponding event of π and $h(e_1 \bullet) = h(e'_1 \bullet)$. Then $e'_1 \triangleright e_2 \Rightarrow h(e_1) <_{tar} h(e_2)$.

Proof. By $e'_1 \triangleright e_2$, we know that $e'_1 \in \text{Max}(\bullet(\bullet e_2))$, and therefore $e'_1 < e_2$ and $[e'_1] \subseteq [e_2] \setminus \{e_2\}$. By Lemma 3, we can construct a sequence of events g_1, g_2, \dots, g_k such that $[e'_1] = [e_2] \setminus \{e_2\} \setminus \{g_k\} \setminus \{g_{k-1}\} \setminus \dots \setminus \{g_1\}$. Let $h(g_1), \dots, h(g_k)$ be denoted as t_1, \dots, t_k . By recursively applying Lemma 3 we know $\text{Mark}([e'_1]) \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_k} \text{Mark}([e_2] \setminus \{e_2\})$. From $\text{Mark}([e_1]) = \text{Mark}([e'_1])$ we know that $\text{Mark}([e_1]) \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_k} \text{Mark}([e_2] \setminus \{e_2\})$. Since $e'_1 \triangleright e_2$, only tokens in $\text{Cut}([e'_1]) \setminus e'_1 \bullet$ are required to enable the previous firing sequence. Because of this and $h(e'_1 \bullet) = h(e_1 \bullet)$, it can be inferred that $\text{Mark}([e_1] \setminus \{e_1\}) \xrightarrow{t_1} M'_1 \xrightarrow{t_2} \dots \xrightarrow{t_k} M'_k \xrightarrow{h(e_1)} \text{Mark}([e_2] \setminus \{e_2\})$. And therefore $h(e_1) <_{\text{tar}} h(e_2)$.

With the above rules, we may quickly confirm $t_1 <_{\text{tar}} t_2$ when $t_1 \bullet \cap \bullet t_2 \neq \emptyset$ (if it holds) if we find some event pairs related to t_1, t_2 satisfy the conditions of Prop. 6 and 7, which takes much less time to confirm than the cut enumeration check in Algorithm 1. The following algorithm implements this idea. It is named Early-Confirm Check because of its possibility to confirm TAR at the early stage of the checking based on Prop. 6 and Prop. 7.

Algorithm 3 Early-Confirm Check for TAR by Structural Relations

Input: The original net S and its CPU π , transitions t_1, t_2 .

Output: Returns true if $t_1 <_{\text{tar}} t_2$ is confirmed, false if unconfirmed yet.

```

1: function EARLYCONFIRM( $S, \pi, t_1, t_2$ )
2:   for each  $e_1 : h(e_1) = t_1$  do
3:     if  $e_1$  is a cut-off event then
4:        $e'_1 \leftarrow$  the non-cut-off corr. event of  $e_1$ 
5:       for each  $e_2 : h(e_2) = t_2$  do
6:         if  $h(e'_1 \bullet) = h(e \bullet)$  and  $e'_1 \triangleright e_2$  then return true
7:       else
8:         for each  $e_2 : h(e_2) = t_2$  do
9:           if  $e_1 \triangleright e_2$  then return true
10:      end if
11:    end for
12:  return false
13: end function

```

Algorithm 3 can confirm TARs quickly except for cases when TAR of a pair of consecutive transitions ($t_1 \bullet \cap \bullet t_2 \neq \emptyset$) cannot be confirmed using the above MEA structures. In the following Fig. 5, we show such a special example, where additional checking using Algorithm 1 is necessary to confirm TAR.

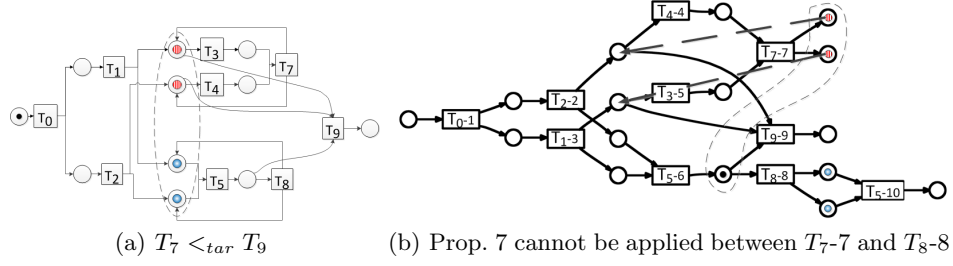


Fig. 5: An artificial special example in which MEA cannot be applied

In the artificial model as shown in Fig. 5, the TAR between the consecutive transitions T_7 and T_9 cannot be confirmed directly using MEA, because T_{7-7} is a *cut-off* event. Moreover, neither can we discover this TAR by reusing the MEA related to T_{7-7} 's corresponding event T_{8-8} based on Prop. 7, because Prop. 7 requires that the corresponding places of the post conditions of the two events are identical, but in this case $h(T_{7-7}\bullet) \neq h(T_{8-8}\bullet)$.

But for such pairs of transitions, Algorithm 1 still applies because it does not require the MEA structures of the related events, so we can combine the boosting rules (Algorithm 3) with Algorithm 1 for accelerated and general TAR checking. All in all, this final improved algorithm for finding all TARs from the CPU (π) of a Petri net (S) can be given as Algorithm 4:

Algorithm 4 Improved Find-All-TAR Function (IMPROVED)

```

1: function IMPROVED( $S, \pi$ )
2:    $TAR \leftarrow \emptyset$ 
3:   for each  $t_1 \in S$  do
4:     for each  $t_2 \in (t_1\bullet)\bullet$  do
5:       if EarlyConfirm( $t_1, t_2$ ) then  $TAR \leftarrow TAR \cup \{ \langle t_1, t_2 \rangle \}$ 
6:       else if CheckByCuts( $t_1, t_2$ ) then  $TAR \leftarrow TAR \cup \{ \langle t_1, t_2 \rangle \}$ 
7:   for each  $e_1, e_2 \in \pi : e_1 \text{ co } e_2$  do
8:      $TAR \leftarrow TAR \cup \{ \langle h(e_1), h(e_2) \rangle \} \cup \{ \langle h(e_2), h(e_1) \rangle \}$ 
9:   return  $TAR$ 
10: end function

```

Proposition 8. *Algorithm 4 can find all TARs of any given bounded net system.*

Proof. The proof is immediate from the Prop. 5, 6, 7 and the correctness of Algorithm 1 (CheckByCuts), which is given by Proposition 3.

5 Experiment Evaluation

We conduct experiments to evaluate the effectiveness of our CPU-based algorithms using both industrial (see [12] for further details) and artificial process model datasets. Implementations of algorithms and the process model datasets

have been made available on-line¹. All experiments were run on a PC with Intel Dual Core I5 CPU@2.8G, 4G DDR3@1333MHz, and Windows 7 Enterprise OS.

5.1 Effectiveness Evaluation

Table 1: Features of the five process model libraries

	A2	B2	B3	B4	M1
Bounded	256	393	338	275	27
Avg./Max Concurrency	2/13	8/14	16/66	14/33	2/4
Time-Limit Exceeded for RG	0	39	36	24	3

Table 1 summarizes the statistics of the our experiment dataset consisting of 5 libraries, from which we extract the bounded models for our experiment. The table also shows the degree of concurrency found in the process models, i.e. the maximum number of tokens that occur in a single reachable non-error state of the process.

First, using the bounded nets in each library, we compare the time cost (see Fig. 6(a)) of two CPU-based algorithms, i.e. GENERAL (Algorithm 2) and Jin’s limited CPU-based approach (JIN) [10], and two RG based algorithms (pure RG based algorithm in [9] (RG) and Zha’s improved RG based approach (ZHA) [1]). Time costs of building CPUs alone in each library (CPU-BUILD) are also shown in the figure along with the time cost of these algorithms. Since RG based methods may encounter state-explosion when applied on nets with high concurrencies, for cases where the RG of a net system cannot be constructed within acceptable time limit, we mark the target net as ”Time Limit Exceeded for RG”. It can be observed that as nets in B2, B3 and B4 contain more concurrent structures, GENERAL performs significantly better than all RG based approaches while only being a bit slower than RG in A2, M1, which can be explained by the lacking of concurrent structures in these two libraries. From this figure we can also see that whereas JIN only supports free-choice nets, our GENERAL achieves a much greater generality (supports any bounded net system) at the expense of only a bit more overhead and shows comparable performance with JIN. Furthermore, we observe the boosting effect of the Algorithm 4 (IMPROVED) against GENERAL (Fig. 6(b)). We compute the overhead of IMPROVED and GENERAL, i.e., their total TAR derivation time cost minus CPU-BUILD time, which costs the same for both algorithms. From Fig. 6(b), we observe that the total overheads for GENERAL to derive TAR from CPUs are around 7 times larger than IMPROVED, which, as a result, causes IMPROVED to run much faster than GENERAL.

As an illustration of the effectiveness of the proposed algorithms on non-free-choice structures, in Table 2 we compare TAR results derived from the non-free-choice net as shown in Fig. 2(a) of Section 3.1 by the above algorithms. As GENERAL, IMPROVED and RG based algorithms get all correct TARs, the method in [10] (JIN) outputs two wrong TARs ($T_0 <_{tar} T_3, T_1 <_{tar} T_4$).

¹ <http://laudms.thss.tsinghua.edu.cn/trac/Test/raw-attachment/wiki/Share/CTAR.zip>

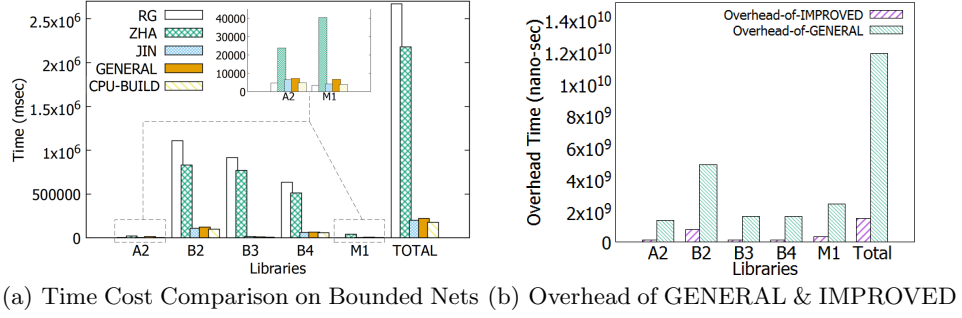


Fig. 6: Effectiveness Evaluation of Proposed TAR Computing Algorithms

According to the rule in [10], as T_0-1 is connected to T_3-5 directly with a condition in the related CPU (see Fig. 3), T_0 is judged to be in TAR with T_3 , but such deduction only applies to free-choice models and is wrong for non-free choice nets. The same mistake also happens for the case of T_1 and T_4 . This shows that the rule used in [10] does not apply to non-free-choice structures, whereas the correctness of proposed GENERAL and IMPROVED algorithms with such structures have already been proved for any bounded net system.

Table 2: TAR Results of Non-free-choice Net Example in Fig. 2(a)

Approaches	TAR Results				
RG, ZHA	$\langle T_0, T_2 \rangle$	$\langle T_1, T_2 \rangle$	$\langle T_2, T_3 \rangle$	$\langle T_2, T_4 \rangle$	
GENERAL, IMPROVED	$\langle T_0, T_2 \rangle$	$\langle T_1, T_2 \rangle$	$\langle T_2, T_3 \rangle$	$\langle T_2, T_4 \rangle$	
JIN	$\langle T_0, T_2 \rangle$	$\langle T_1, T_2 \rangle$	$\langle T_2, T_3 \rangle$	$\langle T_2, T_4 \rangle$	$\langle T_0, T_3 \rangle \langle T_1, T_4 \rangle$

5.2 Scalability Evaluation

We compare the scalability of GENERAL, IMPROVED with existing JIN, ZHA and RG on artificial Petri nets with growing concurrency structures. In Test A, each net is a single AND-split structure ended with an AND-join with exactly one transition on each branch, and we observe how the increase in the number of concurrent branches affects TAR computation time. In Test B, we generate concurrent branches of a fixed number (e.g. 5) and observe the effect of the number of transitions on each branch on the performance of algorithms. In both tests, CPU-based algorithms (GENERAL, IMPROVED, JIN) scale much better than RG based algorithms (RG, ZHA). But JIN does not support non-free-choice nets whereas GENERAL and IMPROVED do. Among CPU-based algorithms, IMPROVED scales the best with the growing size of concurrency structures, which is more efficient than GENERAL in both tests and has comparable or even better performance than JIN.

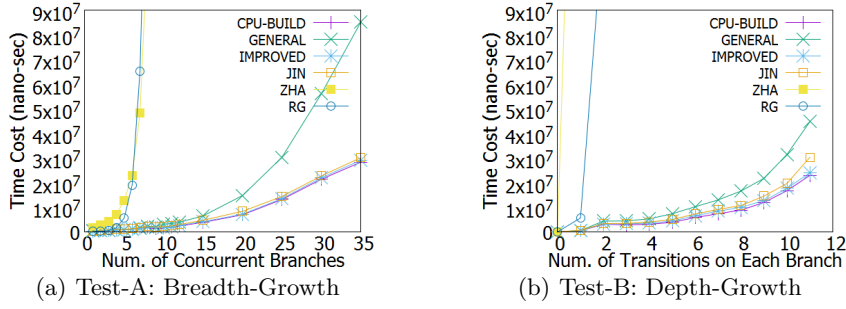


Fig. 7: Scalability Evaluation of TAR Computing Algorithms

6 Conclusion and Future Work

In this paper, we investigated the problem of deriving TARs from the CPU of a Petri net. We propose and prove TAR-EAR relationship to capture the manifestation of TAR in CPU. We put forward the concept of Max-Event Set and the three important lemmas related with this concept, based on which a general algorithm for computing TARs from CPU is proposed. Moreover, by utilizing structural relations, i.e. concurrency and the novel concept of Max-Event Adjacent structures, we further improved the efficiency of the general algorithm. The formal discussion and proof of generality and correctness of these CPU-based improvements are also provided in this work for the first time. Experiment results indicate CPU-based approaches are well-suited for handling the complexities of Petri net while preserving the generality, and also that the improved algorithm significantly reduces the overhead compared to the general algorithm.

As future work, we want to explore the possibilities of taking silent tasks into consideration and adding more accelerations to our existing algorithms.

References

1. Zha, Haiping, et al. "A workflow net similarity measure based on transition adjacency relations." *Computers in Industry* 61.5 (2010): 463-471.
2. Weidlich, Matthias, Jan Mendling, and Mathias Weske. "Efficient consistency measurement based on behavioral profiles of process models." *Software Engineering, IEEE Transactions on* 37.3 (2011): 410-429.
3. Mac Millan, K. "A technique of state space search based on unfolding." *Journal of Formal Methods and System Design* 9 (1992): 1-22.
4. Esparza, Javier, Stefan Romer, and Walter Vogler. "An improvement of McMillan's unfolding algorithm." *Formal Methods in System Design* 20.3 (2002): 285-310.
5. Rozinat, Anne, and Wil MP van der Aalst. "Conformance checking of processes based on monitoring real behavior." *Information Systems* 33.1 (2008): 64-95.
6. van der Aalst, Wil MP. "Inheritance of business processes: A journey visiting four notorious problems." *Petri Net Technology for Communication-Based Systems*. Springer Berlin Heidelberg, 2003. 383-408.

7. Dumas, Marlon, Luciano Garcia-Banuelos, and Remco M. Dijkman. "Similarity Search of Business Process Models." *IEEE Data Eng. Bull.* 32.3 (2009): 23-28.
8. Van Der Aalst, Wil. *Process mining: discovery, conformance and enhancement of business processes*. Springer Science & Business Media, 2011.
9. Prescher, Johannes, Jan Mendling, and Matthias Weidlich. "The Projected TAR and its Application to Conformance Checking." *EMISA*. 2012.
10. Jin, T., Wang J., and Wen, L. "Efficient retrieval of similar workflow models based on behavior." *Web Technologies and Applications*. Springer Berlin Heidelberg, 2012. 677-684.
11. Weidlich, M., Felix Elliger, and Mathias Weske. "Generalised computation of behavioural profiles based on petri-net unfoldings." *Web Services and Formal Methods*. Springer Berlin Heidelberg, 2011. 101-115.
12. Fahland, Dirk, et al. "Instantaneous soundness checking of industrial business process models." *Business Process Management*. Springer Berlin Heidelberg, 2009. 278-293.
13. Murata, T.: *Petri nets: Properties, analysis and applications*. *Proceedings of the IEEE* 77(4), 541-580 (1989)